International Journal of Science and Research (IJSR)
ISSN: 2319-7064
ResearchGate Impact Factor (2018): 0.28 | SJIF (2018): 7.426

# Decoding the Java Toolkit: An Insightful Analysis of Frameworks for Large - Scale Distributed Applications

**Mahidhar Mullapudi[1], Lakshmi Mullapudi[2], Mounika Gorantla[3]**

**Abstract:** ***This research paper presents an in - depth examination of the Java toolkit available at the time of writing this paper, focusing on its significance in the development of large - scale distributed applications and microservices. Key components such as Java 8, Angular4, HTML5, Bootstrap, CSS, Flex, Spring Boot, and Spring Security, Spring Data, SQL and NoSQL databases, hibernate etc., are rigorously evaluated based on critical metrics such as performance, scalability, and ease of use [1] [2] [3] [4]. Our study extends its purview to DevOps practices, investigating streamlined build and release pipelines. Additionally, it explores the integration of cloud infrastructure for efficient deployments on leading platforms such as AWS and Azure. Within the context of microservices, essential terms such as service discovery, containerization, and orchestration are introduced to underscore their role in modern distributed systems [5]. Highlighting the central role of microservices, the paper intricately details how Spring Boot facilitates the creation and maintenance of these modular components. Renowned for its lightweight and opinionated framework, Spring Boot streamlines the development and deployment of microservices, addressing complexities through embedded containers and out - of - the - box support for crucial features like service discovery[6][7][8]. In the realm of end - to - end testing, Selenium WebDriver emerges as the preferred choice, seamlessly integrating into the Java stack, including Spring Boot and associated modules. Coupled with JUnit 5[9] and Mockito[10], these tools enhance testing strategies for a comprehensive approach[11]. With years of experience and practical utilization of some of these tools in the Production environment and thorough research serves as a valuable guide for practitioners navigating the intricacies of building robust, scalable, and cloud - native distributed systems, with a specific emphasis on microservices, at the time of writing this paper. We believe the insights presented herein align with the caliber of research sought by esteemed publishers, contributing meaningfully to the discourse on contemporary software development practices [12][8][13][14][15].***

**Keywords:** Modern Software Applications, Java application stack, Spring Boot, Angular, Cloud computing, DevOps best practices, Microservice architectures, Service Discovery.

## 1. Introduction

In the dynamic realm of software engineering, where architecture and tooling choices profoundly impact the success of large - scale distributed applications, this research embarks on an exhaustive exploration of the Java toolkit available at the time of writing this paper. The decision - making process for selecting development libraries is a critical juncture, and this paper aims to unravel the complexities of why certain libraries were chosen over others. The guiding principles in these selections revolve around the nuanced considerations of performance, scalability, ease of use, and community support. Beyond mere technological preferences, this paper delves into the strategic advantages each library brings to the table and the considerations that propelled them to the forefront of our toolkit recommendations.

**Key Pointers Considered**:
In proposing the adoption of specific libraries, paramount considerations included:

- **Performance Excellence**: A meticulous evaluation of how each library contributes to the overall performance of the application.
- **Scalability Features**: An assessment of the scalability features inherent in the chosen libraries to accommodate the potential growth of the application[16].
- **Ease of Use and Integration**: The seamless integration of libraries within the existing stack, coupled with an emphasis on developer - friendly practices and minimal learning curves.
- **Robust Community Support**: The vitality of a thriving community for ongoing support, updates, and collaborative problem - solving.
- While proposing the adoption of these libraries and the architectural paradigms surrounding them, success metrics were identified to gauge the efficacy of the chosen approach. These metrics include:
- **System Performance**: Improvements in application performance metrics, such as response time, throughput, resource utilization.
- **Scalability**: Assessing the system's ability to scale efficiently with increased loads.

| Stack Components | Technologies |
|---|---|
| Frontend UI | JavaScript, HTML, CSS |
| Frontend Libraries | React, Angular |
| Front End styling | Bootstrap, Material Design |
| Programming | Java, Node, Python |
| Web frameworks | Spring, Django |
| Database | NoSQL - MongoDB<br>SQL - Oracle, MySQL |
| Event and messaging | Kafka |
| Infrastructure | Azure, AWS, Google Cloud |
| Virtualization | Kubernetes, Docker |

- **Developer Productivity**: Measuring the impact on developer productivity through metrics like reduced development time, ease of debugging, and efficient collaboration.
- **Community Engagement**: Monitoring the engagement and contributions within the community surrounding the selected libraries.

- **Operational Efficiency**: Evaluating the efficiency gains in operational processes, including deployment, maintenance, and troubleshooting.

The paper is organized into distinct sections to facilitate comprehensive exploration. The system overview delineates the roles and interactions of each development library, setting the stage for an in - depth examination. The subsequent deep dive sections dive into each library individually, presenting insights into their advantages and areas of excellence. Following this, the paper navigates through the evolving landscape of microservices architecture, the practices of DevOps[5], and the pivotal role of cloud deployments[6]. Each section contributes to a holistic understanding of the Java toolkit, empowering readers to navigate the intricate choices inherent in building large - scale distributed applications.

## 2. Systems Overview

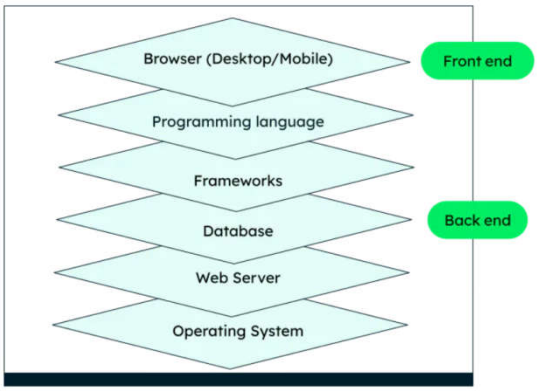


**Figure 1:** Tech Stack for a Web Application

As illustrated in Figure 1, the tech stack for a web application consists of different layers. A stack is an arrangement of "things" kept in order one over the other [12]. Tech stacks are sets of technologies that are stacked together to build any application. Popularly known as a technology infrastructure or solutions stack, tech stacks have become essential for building web applications that are easy to maintain and scalable. Tech stacks determine the type of applications you can build, the level of customizations you can perform, and the resources you need to develop your application.

The table above outlines the list of components in a tech stack and corresponding technologies.

In the intricate landscape of large - scale distributed applications, the system's architecture is a tapestry woven from a multitude of components, each playing a distinct role. This section provides a panoramic introduction to the diverse components that constitute the Java toolkit, carefully grouped based on their roles in shaping the holistic system. The symphony of frontend, backend, testing, logging components, along with layers dedicated to build, release, and deployments, and the core infrastructure, collaboratively crafts a robust ecosystem. As we embark on this exploration, envision how each component seamlessly integrates into the larger framework, contributing to the cohesive and efficient operation of the entire system.

**Frontend & UI Layer**: At the forefront of user interaction, the Frontend & UI Layer orchestrates a captivating user experience. Angular 4 [17] spearheads the frontend design using TypeScript, complemented by the versatility of HTML5[1][2], the responsive styling of Bootstrap[18], and the expressive power of CSS[19][20] and Flex. Together, these components lay the foundation for responsive applications.

**Backend & Services:**
Beneath the surface, the Backend & Services layer forms the backbone of the system, responsible for handling business logic and data operations. Java 8 assumes the mantle for overall development, while Spring Boot takes center stage in wiring and orchestrating the diverse components. Spring Security fortifies the security layer, ensuring robust protection against potential threats, and the Spring Dependency Injection optimizes the management of dependencies through Inversion of Control (IOC). Spring REST and Spring Data modules power the services, offering seamless integration and efficient data handling. Databases, represented by MongoDB [21] for NoSQL needs and a choice between Oracle or PostgreSQL [22] for SQL requirements, serve as repositories of structured information. Redis, functioning as the caching layer, optimizes data retrieval and enhances overall performance.

**Logging & Testing:**
In the quest for maintainability and reliability, the Logging & Testing layer provides essential tools for developers. SLF4J [23]and Log4j [24] collaborate to capture and organize log information, offering insights into the system's operational health. The Testing suite, fortified by JUnit 5, Mockito, and JMeter, ensures the robustness and efficiency of the application. From unit tests to performance evaluations, these tools contribute to a comprehensive testing strategy [25].

**Build, Release & Deployments**:
Beyond the development and testing phases, the Build, Release & Deployments layer governs the systematic progression of the application toward production. Jenkins, Maven, and Docker collaborate to facilitate continuous integration and continuous deployment (CI/CD) processes, ensuring a seamless transition from code changes to live apps [26].

**Core Infrastructure:**
Ensuring the seamless operation of the entire system, the Core Infrastructure layer encompasses cloud deployments and fundamental services. Basic cloud services form the bedrock for scalability and efficiency, and the integration with Microsoft Kubernetes service, provides orchestration for containerized applications. This layer plays a pivotal role in scaling the application horizontally and vertically, adapting to evolving demands [5][6].

As these components and layers interweave, they transcend their roles when woven together into the fabric of the overall system. The frontend interfaces seamlessly with the backend, facilitated by the orchestrated efforts of Spring Boot. Security concerns are addressed by Spring

Security[27], while dependency injection ensures modular and maintainable code. Data flows seamlessly between databases, and caching optimizes resource utilization. Logging mechanisms capture valuable insights, testing strategies validate the system's resilience, and build, release, and deployment processes ensure a streamlined life cycle. The core infrastructure, with its emphasis on cloud deployments and fundamental services, adds the final layer of robustness to the overarching architecture. In the subsequent sections, this paper will delve into a more granular examination of each component, unraveling their advantages, strategic considerations, and potential rivals. We also explore microservices architecture, DevOps practices, and cloud deployments in the subsequent sections.

## 3. Deep Dive into the Tech Stack

**User Interfaces**:
The User Interfaces layer, a critical facet of any large - scale application, is meticulously crafted through a combination of Angular 4, HTML5, Bootstrap, CSS, and Flex.

**Angular 4**: Angular 4[17] emerges as the frontrunner in our UI arsenal due to its comprehensive framework and forward - thinking features. The extensive use of TypeScript enhances maintainability, and Angular's two - way data binding simplifies complex UI updates. Its powerful CLI streamlines project scaffolding and build processes. In comparison to React[28], Angular provides a more structured and opinionated approach, minimizing decision fatigue to enhance developer productivity.

*Sample Code:*

```
// Angular Component
import { Component } from'[at]angular/core';
[at]Component ({
 selector: 'app - root',
 template: '<h1>Hello {{name}}</h1>',
})
exportclassAppComponent {
 name = 'Angular 4';
}
```

**HTML5, CSS, Flex**: This amalgamation of HTML5, CSS, and Flex ensures responsive and visually compelling interfaces. HTML5's semantic markup provides a foundation for accessibility and search engine optimization.

**Bootstrap:** Powerful, extensible, and feature packed frontend toolkit. Build and customize with Sass, utilize prebuilt grid systems and components, and bring projects to life with powerful JavaScript plugins. Bootstrap's grid system, coupled with CSS for styling and Flex for flexible design, establishes a harmonious visual language [18][29].

**Java 8**: The adoption of Java 8 introduces a paradigm shift in development practices, leveraging lambdas, streams, and the java. Time package. This enhances code conciseness and readability, especially in scenarios demanding asynchronous programming or functional transformations. Java 8's rivaled predecessor, Java 7, falls short in providing these language enhancements[4][14].

```
// Java 8 Lambda Expression
List<String> names = Arrays. asList ("John", "Jane", "Doe");
names. forEach (name - >System. out. println ("Hello, " + name));
```

**Spring Boot**:
The cornerstone of our application, Spring Boot, embodies a convention - over - configuration philosophy, simplifying development tasks and promoting modular architecture. Its embedded container, auto - configuration, and minimal setup allow developers to focus on business logic. In contrast to rivals like Struts, Play Framework, Spring Boot's broad ecosystem, community support, and seamless integration make it our architectural linchpin[30][7].

*Code Sample for Spring Boot Application Class*:

```
[at]SpringBootApplication
publicclassMyApplication {
publicstaticvoid main (String [] args) {
SpringApplication. run (MyApplication. class, args);
 }
}
```

**Spring Security**:
Spring Security stands tall as the go - to solution for building robust and customizable security layers. Its extensibility and adaptability outshine alternatives such as Apache Shiro or Java EE Security, providing a comprehensive suite for authentication and authorization.

```
[at]Configuration
[at]EnableWebSecurity
public class SecurityConfig extends WebSecurityConfigurerAdapter {
[at]Override
 protected void configure (HttpSecurity http) throws Exception {
http
. authorizeRequests ()
. antMatchers ("/public/**"). permitAll ()
. anyRequest (). authenticated ()
. and ()
. formLogin (). loginPage ("/login"). permitAll ()
. and ()
. logout (). permitAll ();
 }
}
```

**Spring Dependency Injection**:
The heart of Spring's Inversion of Control, Spring DI, enables loose coupling and testability. Its advanced features like auto wiring and component scanning reduce boilerplate code. Compared to rivals like Google Guice or Java EE CDI, Spring DI's broad feature set, extensive documentation, and mature ecosystem make it the preferred choice[31].
*Code Sample for Spring Dependency Injection:*

```
[at]Component
publicclassMyService {
privatefinalMyRepository repository;
```

```
[at]Autowired
publicMyService (MyRepository repository) {
this. repository = repository;
}
}
```

**Spring REST, Spring Data**:
Spring REST and Spring Data, integral components of our API and data access layers, respectively, facilitate seamless development. Spring REST simplifies API creation, while Spring Data provides a unified interface for interacting with various data sources. Alternatives like Jersey lack cohesive integration and a convention - based approach[32].

```
[at]RestController
[at]RequestMapping ("/api")
publicclassMyController {
[at]GetMapping ("/greet")
public String greet () {
return"Hello, World!";
}
}
```

**MongoDB**:
Chosen for its flexible schema and scalability, MongoDB embodies the NoSQL paradigm. Its JSON - like document store and dynamic schema accommodate evolving data structures. While CouchDB and others offer similar features, MongoDB's wide adoption, developer - friendly queries, and robust community support position it as our NoSQL database of choice[25][21].

```
[at]Repository
publicinterfaceUserRepositoryextendsMongoRe
pository<User, String> {
// Custom queries. . .
}
```

**Oracle/PostgreSQL**:
The relational databases Oracle and PostgreSQL cater to specific project requirements. Oracle, with its mature feature set and robustness, is preferred for enterprise - grade applications. PostgreSQL, being open - source and extensible, excels in scenarios demanding flexibility and community - driven innovation. Both surpass competitors in terms of reliability, scalability, and ACID compliance[33].

```
// Spring Data JPA Repository
[at]Repository
publicinterfaceUserRepositoryextendsJpaReposit
ory<User, Long> {
// Custom queries. . .
}
```

**Redis**:
Redis, our caching layer, provides in - memory data storage and versatility. Its support for complex data structures and efficient caching mechanisms surpasses alternatives like Memcached. Redis' persistence features and ease of use contribute to its standing as a caching powerhouse.

```
// Redis Cache Configuration
[at]Configuration
[at]EnableCaching
public classCacheConfigextendsCachingConfigurerSuppor
t {
[at]Bean
public RedisCacheManagercacheManager
(RedisConnectionFactoryconnectionFactory) {
RedisCacheConfiguration config =
RedisCacheConfiguration. defaultCacheConfig ()
. entryTtl (Duration. ofMinutes (5))
. disableCachingNullValues ();

returnRedisCacheManager. builder
(connectionFactory)
. cacheDefaults (config)
. transactionAware ()
. build ();
}
}
```

**Testing and Best Practices**:
**JUnit 5 and Mockito**:
Our testing suite embraces JUnit 5 for its flexibility and extensibility. Mockito, in tandem with JUnit 5, simplifies mocking for effective unit testing. While alternatives like TestNG or Easy Mock serve specific needs, JUnit 5 and Mockito shine in terms of community support, documentation, and seamless integration[9][10].

```
// JUnit 5 Test Case
[at]Test
publicclassMyServiceTest {
[at]Test
voidmyServiceShouldReturnExpectedValue () {
MyServicemyService = newMyService ();
String result = myService. doSomething ();
assertEquals ("ExpectedValue", result);
}
}
```

**JMeter**:
Our choice for performance testing, JMeter, boasts versatility and scalability. While alternatives like Gatling or Locust may excel in specific scenarios, JMeter's comprehensive feature set, extensive user base, and user - friendly interface make it the preferred tool for holistic performance evaluations.

By deeply scrutinizing each component within the tech stack, backed by thorough comparisons and code samples, this research paper takes an opinionated approach in selecting the libraries for web application development. The chosen libraries and frameworks are not merely asserted but substantiated by tangible evidence, providing readers with a robust foundation for informed decision - making in designing and building large - scale distributed application development.

## 4. Microservice Architecture

Microservices, a paradigm in software architecture, advocates for building applications as a collection of small, independent services, each focused on a specific business capability. This section briefly introduces key concepts and

patterns in microservice architecture and elucidates how Spring Boot, with its inherent design philosophy and features, facilitates the creation and maintenance of microservices[32][8][34].

**Key Concepts and Patterns**:
**Service Independence**: Microservices operate as independent entities, enabling teams to develop, deploy, and scale services independently, fostering flexibility and agility.

**Decentralized Data**: Eachmicroservice manages its own data storage, minimizing dependencies and simplifying data management.

**API Gateway**: An API gateway aggregates and exposes APIs, simplifying client communication and providing a single - entry point.

**Service Discovery**: Services dynamically discover and communicate with each other, ensuring seamless integration and adaptability.

**Fault Tolerance and Resilience**: Microservices embrace resilience patterns like circuit breakers to ensure system robustness and availability.

**How Spring Boot Enables Microservices**:
Spring Boot, renowned for simplifying Java development, provides a conducive environment for building and maintaining microservices. Its features streamline the development process, ensuring a cohesive and efficient microservices architecture.

```
// Spring MVC Controller for User Service
[at]RestController
[at]RequestMapping ("/users")
publicclassUserController {
[at]Autowired
privateUserServiceuserService;

[at]GetMapping ("/{userId}")
publicResponseEntity<User>getUserById
([at]PathVariable String userId) {
 User user = userService. getUserById (userId);
returnResponseEntity. ok (user);
 }

// Other CRUD operations. . .
}

// Eureka Server Configuration
[at]EnableEurekaServer
[at]SpringBootApplication
public class EurekaServerApplication {
publicstaticvoidmain (String [] args) {
SpringApplication. run
(EurekaServerApplication. class, args);
 }
}

// Eureka Client Configuration
[at]EnableDiscoveryClient
[at]SpringBootApplication
public class UserServiceApplication {
publicstaticvoidmain (String [] args) {
SpringApplication. run (UserServiceApplication.
class, args);
 }
}
```

These code samples provide a glimpse into how Spring Boot[7][32][30] simplifies the creation of microservices. The annotations, auto - configuration, and seamless integration with tools like Eureka showcase the efficiency and clarity Spring Boot brings to microservices development. The subsequent sections will delve deeper into DevOps practices and cloud deployments, contributing to a comprehensive understanding of building scalable and resilient distributed systems.

## 5. Devops and Cloud Deployments

DevOps practices and cloud deployments are integral components of modern software development, fostering collaboration, automation, and scalability. This section explores the tools proposed earlier and outlines best practices for a seamless DevOps pipeline and cloud deployment.

**Continuous Integration (CI):** Embrace tools like Jenkins to automate code integration and validation, ensuring that changes are continuously tested and integrated into the main codebase.

**Continuous Deployment (CD):** Maven, integrated with Jenkins, facilitates continuous deployment by automating the packaging and deployment process, enabling swift and reliable releases.

**Containerization with Docker**: Docker simplifies application deployment by encapsulating applications and their dependencies into containers. Docker, combined with Docker Compose, ensures consistency across various environments.

**Kubernetes for Container Orchestration**[35]: Kubernetes provides a robust platform for automating the deployment, scaling, and management of containerized applications. Kubernetes enables efficient orchestration, ensuring high availability and reliability.

**Cloud Deployments**:

**Basic Cloud Services**: Leverage foundational cloud services for scalable and efficient infrastructure. Cloud providers like AWS, Azure, or Google Cloud offer a range of services for computing, storage, and networking.

**Microsoft Kubernetes Service (AKS):** Utilize Kubernetes for container orchestration, providing automated deployment, scaling, and management of containerized applications. Microsoft Azure Kubernetes Service (AKS) streamlines Kubernetes deployment on Azure[5].

**Best Practices**:

**Infrastructure as Code**: Use tools like Terraform or Azure Resource Manager (ARM) templates to define and manage infrastructure as code, enabling reproducibility and versioning.

**Immutable Infrastructure**: Adopt the practice of treating infrastructure as immutable, where servers and components are replaced rather than updated. This ensures consistency and minimizes configuration drift.
**Microservices Scalability**: Leverage cloud - native features for microservices scalability. Autoscaling, load balancing, and serverless architectures optimize resource utilization and enhance system responsiveness [34].

**Monitoring &Logging**: Implement comprehensive monitoring and logging using tools like Prometheus, Grafana, or Azure Monitor. Proactively monitor performance, detect issues, and ensure robust system health [36][37][38].

These best practices, when applied in conjunction with the proposed tools, contribute to a streamlined and efficient DevOps pipeline and cloud deployment strategy. The subsequent section will conclude the research paper with a comprehensive summary and insights into the discussed architectural and operational strategies.

## 6. Conclusion

In conclusion, this research paper has delved into the intricacies of building large - scale distributed software applications using a carefully curated stack of tools and frameworks.
The selection process, guided by the principles of efficiency, scalability, and maintainability, has resulted in a comprehensive tech stack:

- Java 8 for overall development
- Angular 4 for frontend
- HTML5, Bootstrap, CSSfor user interfaces
- Spring Boot for overall wiring
- Spring Security for the security layer
- Spring Dependency Injection for Inversion of Control and Dependency Management
- Spring REST, Spring Data[39] for services
- MongoDB as a NoSQL database
- Oracle or PostgreSQL as SQL databases
- Redis for the caching layer
- SLF4j and Log4j for logging
- JUnit 5 and Mockito for testing
- JMeter for performance testing

Additionally, the adoption of DevOps practices, containerization with Docker, orchestration with Kubernetes, and cloud deployment on platforms like Microsoft Azure Kubernetes Service (AKS) contribute to an end - to - end solution for building robust and scalable software systems.

The embrace of microservices architecture, facilitated by Spring Boot, introduces flexibility and agility into the development process. The selected tools and frameworks seamlessly integrate, providing a cohesive environment for creating, testing, and deploying microservices. Throughout this paper, code samples have been provided to illustrate the practical implementation of the proposed technologies. These samples serve as concrete examples, aiding developers in understanding the syntax and usage of each tool in the specified context.

## References

[1] "Html5 Wiki, " [Online]. Available: https: //en. wikipedia. org/wiki/HTML5.
[2] "HTML w3 schools, " [Online]. Available: https: //www.w3schools. com/html/.
[3] "What is Java?, " [Online]. Available: https: //azure. microsoft. com/en - us/resources/cloud - computing - dictionary/what - is - java - programming - language.
[4] "Why is Java 8 recommended?, " [Online]. Available: https: //www.java. com/download/why - java - 8 - recommended. html.
[5] "Azure Kubernetes Service (AKS), " [Online]. Available: https: //learn. microsoft. com/en - us/azure/aks/.
[6] "Azure Service Fabric, " [Online]. Available: https: //azure. microsoft. com/en - us/products/service - fabric.
[7] "Building an Application with Spring Boot, " [Online]. Available: https: //spring. io/guides/gs/spring - boot/.
[8] "What are microservices?, " [Online]. Available: https: //microservices. io/.
[9] "JUnit 5, " [Online]. Available: https: //junit. org/junit5/.
[10] "mockito, " [Online]. Available: https: //site. mockito. org/.
[11] "Selenium, " [Online]. Available: https: //www.selenium. dev/.
[12] "What is a Tech Stack and How Do They Work, " [Online]. Available: https: //www.mongodb. com/basics/technology - stack.
[13] "What is Java Spring Boot?, " [Online]. Available: https: //azure. microsoft. com/en - us/resources/cloud - computing - dictionary/what - is - java - spring - boot/.
[14] "What is Java technology and why do I need it?, " [Online]. Available: https: //www.java. com/en/download/help/whatis_java. html.
[15] "What is Microservices Architecture?, " [Online]. Available: https: //cloud. google. com/learn/what - is - microservices - architecture.
[16] Kleppmann, Martin, Designing Data - Intensive Applications, O'Reilly Media, 2017.
[17] "Angular, " [Online]. Available: https: //angular. io/docs.
[18] "Bootstrap 4, " [Online]. Available: https: //getbootstrap. com/docs/4.1/getting - started/introduction/.
[19] "css wiki, " [Online]. Available: https: //en. wikipedia. org/wiki/CSS.
[20] "css w3 schools, " [Online]. Available: https: //www.w3schools. com/Css/.
[21] "MongoDb, " [Online]. Available: https: //www.mongodb. com/.
[22] "PostgreSQL, " [Online]. Available: https: //www.postgresql. org/.
[23] "Slf4j, " [Online]. Available: https: //slf4j. org/.

[24] "Log4j, " [Online]. Available: https: //logging. apache. org/.
[25] "CouchDb, " [Online]. Available: https: //couchdb. apache. org/.
[26] "Docker, " [Online]. Available: https: //www.docker. com/.
[27] "Spring Security Architecture, " [Online]. Available: https: //docs. spring. io/spring - security/reference/servlet/architecture. html.
[28] [Online]. Available: React.
[29] "Bootstrap vs Material Design, " [Online]. Available: https: //stackoverflow. com/questions/50831331/bootstrap - vs - material - ui - for - react.
[30] "Spring Boot wiki, " [Online]. Available: https: //en. wikipedia. org/wiki/Spring_Boot.
[31] "Dependency Injection, " [Online]. Available: https: //docs. spring. io/spring - framework/reference/core/beans/dependencies/factory - collaborators. html.
[32] "Building REST services with Spring, " [Online]. Available: https: //spring. io/guides/tutorials/rest/.
[33] "Oracle SQL Db, " [Online]. Available: https: //docs. oracle. com/en/database/oracle/oracle - database/index. html.
[34] "microservices - best - practices, " [Online]. Available: https: //www.mulesoft. com/sem/lp/whitepaper/api/microservices - best - practices.
[35] "Kubernetes, " [Online]. Available: https: //kubernetes. io/.
[36] "Graphana, " [Online]. Available: https: //grafana. com/.
[37] "Prometheus, " [Online]. Available: https: //prometheus. io/docs/introduction/overview/.
[38] "Prometheus Wiki, " [Online]. Available: https: //en. wikipedia. org/wiki/Prometheus.
[39] "Spring Data JPA, " [Online]. Available: https: //spring. io/projects/spring - data - jpa/.
[40] "What is object graph, " [Online]. Available: https: //stackoverflow. com/questions/2046761/what - is - object - graph - in - java.
[41] "What is JavaScript, " [Online]. Available: https: //developer. mozilla. org/en - US/docs/Learn/JavaScript/First_steps/What_is_JavaScript.
[42] "Typescript, " [Online]. Available: https: //www.typescriptlang. org/.
[43] "Object - Oriented Design and Data Structures - Grpah Traversals, " [Online]. Available: https: //andrewcmyers. github. io/oodds/lecture. html?id=traversals.
[44] "JavaScript Wiki, " [Online]. Available: https: //en. wikipedia. org/wiki/JavaScript.
[45] "Flexbox, " [Online]. Available: https: //css - tricks. com/snippets/css/a - guide - to - flexbox/.
[46] "Browser caching, " [Online]. Available: https: //medium. com/[at]codebyamir/a - web - developers - guide - to - browser - caching - cc41f3b73e7c.
[47] "Apache Kafka, " [Online]. Available: https: //kafka. apache. org/.
[48] "Material Design, " [Online]. Available: https: //m3. material. io/.